Theory of Computation and Complexity

Dr. Faisal Aslam

28-04-2025

Contents

1 Introduction to Space Complexity **3**
1.1 Introduction to Space Complexity 4
1.2 Space Complexity Definitions 4
1.2.1 1. Deterministic Turing Machines (DTM) 4
1.2.2 2. Non-Deterministic Turing Machines (NTM) 4
1.3 Clarifying the Notion of “Visited” in Space Complexity 4
1.3.1 Contrast with Time Complexity 4
1.3.2 Key Implications 5
1.4 Complexity Classes for Space 5
1.4.1 SPACE($f(n)$) 5
1.4.2 NSPACE($f(n)$) 5
1.4.3 PSPACE 5
1.4.4 NPSPACE 6
1.5 Space Complexity of Multitape Turing Machines 6
1.6 Relationship Between Time and Space Complexity 6
1.6.1 Explanation of the Containments 7
1.6.2 Key Observations 7
1.7 Relationship Between NP and PSPACE 7
1.7.1 Closure Under Complementation 8
1.8 Open Problems in Space Complexity 9
1.8.1 Class Containment Problems 10
1.8.2 Specific Separation Problems 10
1.8.3 Space-Time Tradeoffs 11
1.9 The Complexity of TQBF 11
1.9.1 Basic Properties and PSPACE Membership 11
1.9.2 NP-Completeness of TQBF 12
1.9.3 Visualizing the Recursive Evaluation 12
1.9.4 Space Efficiency 13
1.10 The Ladder Problem 13
1.11 Savitch’s Theorem 14
1.12 The Classes L and NL 15
1.12.1 Definitions 15
1.12.2 Key Results 15
1.12.3 Complete Problems 16

1.12.4 Algorithmic Techniques 16
1.12.5 Open Problems 16

Chapter 1

Introduction to Space Complexity

1.1 Introduction to Space Complexity

So far, we have focused on **time complexity**. Today, we will explore **space complexity**, which measures the memory resources an algorithm uses. Space complexity is important for two key reasons:

1. It quantifies the **memory consumption** of an algorithm.

2. If two algorithms have the **same time complexity**, but one uses more space, the latter may run slower due to higher memory overhead.

We will only consider space complexity for **decidable Turing Machines (TMs)**, meaning they halt on all inputs. With this in mind, let's formally define space complexity.

1.2 Space Complexity Definitions

1.2.1 1. Deterministic Turing Machines (DTM)

The **space complexity** of a decidable deterministic TM is a function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the **maximum number of tape cells visited** by the TM on any input of size n.

1.2.2 2. Non-Deterministic Turing Machines (NTM)

The **space complexity** of a decidable non-deterministic TM is a function $f : \mathbb{N} \to \mathbb{N}$, where $f(n)$ is the **maximum number of tape cells visited** by the TM **along any computational branch** for any input of size n.

1.3 Clarifying the Notion of "Visited" in Space Complexity

To properly analyze space complexity, we must precisely define what constitutes a "visited" tape cell:

Definition 1.3.1 (Visited Cell). A tape cell is considered **visited** if the Turing Machine's read/write head ever occupies or scans that cell during computation. Each distinct cell visited contributes exactly one unit to the space complexity, regardless of how many times it is accessed.

1.3.1 Contrast with Time Complexity

- **Space Complexity** counts only the *number of distinct cells* ever accessed during computation. Multiple accesses to the same cell do not increase the space measure.

- **Time Complexity** counts the *total number of transitions* executed by the TM. Each basic operation (read, write, move, state change) constitutes one transition, regardless of whether it involves previously accessed cells.

1.3.2 Key Implications

- Space complexity depends only on the *maximum workspace* needed, not on how frequently cells are reused.
- Time complexity depends on the *total computation length*, counting every transition, including repeated operations on the same cells.
- This distinction explains why some problems can have different space and time complexity classes (e.g., problems in PSPACE but not P).

Example 1.3.1. Consider a TM that writes n bits by repeatedly moving back-and-forth across $O(1)$ cells:

- Its *space complexity* is $O(1)$ (fixed number of cells)
- Its *time complexity* is $\Omega(n)$ (n transitions needed)

1.4 Complexity Classes for Space

We now define some fundamental classes of space complexity:

1.4.1 SPACE($f(n)$)

$\text{SPACE}(f(n)) = \{B \mid \text{some deterministic 1-tape TM } M \text{ decides } B \text{ using } O(f(n)) \text{ space}\}$

A more precise name for this class is **DSPACE($f(n)$)**, but we will use the conventional notation.

1.4.2 NSPACE($f(n)$)

$\text{NSPACE}(f(n)) = \{B \mid \text{some non-deterministic 1-tape TM } M \text{ decides } B \text{ using } O(f(n)) \text{ space}\}$

1.4.3 PSPACE

$$\text{PSPACE} = \bigcup_{k \in \mathbb{N}} \text{SPACE}(n^k)$$

Thus, PSPACE contains all languages decidable by a deterministic TM using **polynomial space**.

1.4.4 NPSPACE

$$\text{NPSPACE} = \bigcup_{k\in\mathbb{N}} \text{NSPACE}(n^k)$$

Thus, NPSPACE contains all languages decidable by a non-deterministic TM using **polynomial space**.

These classes help us categorize problems based on their **memory requirements** under deterministic and non-deterministic computation models.

1.5 Space Complexity of Multitape Turing Machines

Definition 1.5.1. For a k-tape Turing machine M, its **space complexity** is the function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ represents the maximum total number of distinct cells visited across *all* tapes during M's computation on any input of length n.

Theorem 1.5.1 (Multitape to Single-Tape Simulation). Any k-tape Turing machine (TM) with space complexity $O(s(n))$ can be simulated by a single-tape TM with the same space complexity, i.e., $O(s(n))$, with the space only increased by a constant factor. This constant factor depends only on the number of tapes k, and not on the input size n.

1.6 Relationship Between Time and Space Complexity

Theorem 1.6.1 (Time-Space Containment). For any space-constructible function $t(n) \geq n$:

1. $\text{TIME}(t(n)) \subseteq \text{SPACE}(t(n))$
   (A machine using $t(n)$ time can visit at most $t(n)$ tape cells, hence its space usage is bounded by $t(n)$.)

2. $\text{SPACE}(t(n)) \subseteq \text{TIME}(2^{O(t(n))})$
   (A machine using $t(n)$ space has at most $2^{O(t(n))}$ possible configurations, so it must halt within that many transitions if it halts at all.)

3. Based on the first inclusion, we can conclude that:

   - $P \subseteq PSPACE$
   - $EXPTIME \subseteq EXPSPACE$
   - $NP \subseteq NPSPACE$

4. Based on the second inclusion, we can conclude that:

   - $PSPACE \subseteq EXPTIME$

- $NPSPACE \subseteq EXPTIME$
  (Because by Savitch's theorem, $NPSPACE = PSPACE$, so the same time bound applies.)

1.6.1 Explanation of the Containments

Time Bounds Space: Each transition of a TM can access at most one new tape cell. Therefore, a machine that halts after $O(t(n))$ transitions cannot use more than $O(t(n))$ space:

$$\text{TIME}(t(n)) \subseteq \text{SPACE}(t(n))$$

Space Bounds Time via Configurations: For a TM using $O(t(n))$ space:

- Let Q be the set of states ($|Q|$ constant)
- Let Γ be the tape alphabet ($|\Gamma|$ constant)
- Number of head positions: $O(t(n))$
- Number of possible tape contents: $|\Gamma|^{O(t(n))} = 2^{O(t(n))}$

Thus, the total number of distinct configurations is:

$$|Q| \times O(t(n)) \times 2^{O(t(n))} = 2^{O(t(n))}$$

Since the machine must halt (by decidability), it cannot execute more transitions than its number of possible configurations (otherwise it would repeat a configuration and loop forever). Therefore:

$$\text{SPACE}(t(n)) \subseteq \text{TIME}(2^{O(t(n))})$$

1.6.2 Key Observations

- Each transition corresponds to one unit of time complexity
- The configuration count ($2^{O(t(n))}$) provides an upper bound on possible distinct transitions before halting
- This explains why PSPACE $\subseteq$ EXP (problems solvable in polynomial space can require exponential time)

1.7 Relationship Between NP and PSPACE

Theorem 1.7.1. $NP \subseteq PSPACE$

Proof. The proof proceeds in the following steps:

1. **Proof that $SAT \in PSPACE$:** Although SAT is an NP-complete problem, it can be decided using polynomial space by trying all possible truth assignments sequentially without storing all of them simultaneously. We can evaluate each assignment one-by-one using only polynomial space, because storing a single assignment and checking the formula can be done within space proportional to the input size.

2. **Proof that if $A \leq_p B$ and $B \in PSPACE$, then $A \in PSPACE$:** Suppose f is a polynomial-time reduction from A to B, and we have a polynomial-space algorithm for B. To decide if $x \in A$, we compute $f(x)$ (which requires only polynomial time, and hence polynomial space), and then decide whether $f(x) \in B$ using the polynomial-space algorithm for B. Thus, A can be decided in polynomial space.

3. **Conclusion:** Every language $L \in NP$ is polynomial-time reducible to SAT (since SAT is NP-complete). By steps 1 and 2, and since polynomial-time reductions preserve membership in $PSPACE$, we conclude that $L \in PSPACE$. Hence, $NP \subseteq PSPACE$.

□

Theorem 1.7.2. $PSPACE = coPSPACE$

Proof. **1.7.1 Closure Under Complementation**

Theorem 1.7.3 (PSPACE = coPSPACE). The complexity class PSPACE is closed under complementation. That is:

$$\text{PSPACE} = \text{coPSPACE}$$

Proof Sketch. For any language $L \in$ PSPACE, let M be a TM that decides L using polynomial space. We can construct a machine M' that decides $\overline{L}$ (the complement of L) as follows:

1. M' simulates M using the same polynomial space bound $p(n)$
2. When M would accept, M' rejects
3. When M would reject, M' accepts
4. M' maintains the same space complexity since:
   - It uses exactly the same tape cells as M
   - The finite control only needs constant additional space to track the inverted acceptance condition

The key observations are:

- Space-bounded computation can be *deterministically* complemented without increasing space usage

- Unlike time complexity, we don't need to worry about "timing out" - the space bound guarantees termination
- This fails for nondeterministic space classes unless Savitch's theorem gives us determinization (which it does for polynomial space)

□

Contrast with Other Complexity Classes

- **Time Classes:** P = coP (by similar argument), but it remains unknown whether NP = coNP
- **Logarithmic Space:** L = coL (by Szelepcsényi's theorem for NSPACE)
- **General Pattern:** Space-bounded classes tend to be closed under complementation, while time-bounded classes may not be

Significance This property demonstrates an important advantage of space-bounded computation:

- Space resources can be *reused* during computation
- There's no need to store computation history for complementation
- Contrast with time-bounded computation where complementation might require storing or recomputing information

□

Theorem 1.7.4. $coNP \subseteq PSPACE$

Proof. The proof proceeds as follows:

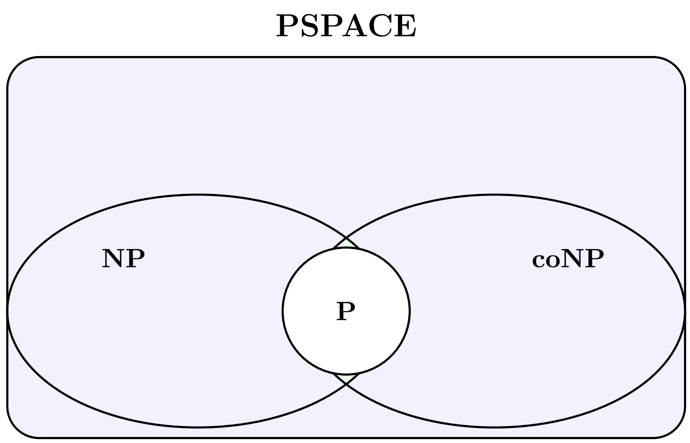
1. **Fact:** It is known that $PSPACE = coPSPACE$, meaning the class of languages decidable in polynomial space is closed under complementation.
2. **Conclusion:** Since $NP \subseteq PSPACE$, taking complements yields $coNP \subseteq coPSPACE = PSPACE$.

□

1.8 Open Problems in Space Complexity

Space complexity theory contains several fundamental unresolved questions. Below we present the most significant open problems with their current status.

Although we know that $\mathbf{NP}, \mathbf{coNP} \subseteq \mathbf{PSPACE}$, it is still unknown whether $\mathbf{NP} = \mathbf{coNP}$ or whether either of them equals $\mathbf{P}$. These inclusions help structure the complexity landscape, as illustrated in the diagram below.



- $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE}$
- $\mathbf{P} \subseteq \mathbf{coNP} \subseteq \mathbf{PSPACE}$
- $\mathbf{NP} \stackrel{?}{=} \mathbf{coNP}$ (Open)
- $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ (Open)

1.8.1 Class Containment Problems

- **P vs PSPACE**

Known: $\mathbf{P} \subseteq \mathbf{PSPACE}$ (via space-time hierarchy)

Open: Is the containment proper?

Significance: Would imply all polynomial-space algorithms can be made time-efficient

Conjecture: $\mathbf{P} \neq \mathbf{PSPACE}$

- **NP vs PSPACE**

Known: $\mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP}$

Open: Is $\mathbf{NP} = \mathbf{PSPACE}$?

Note: Resolution may require new proof techniques beyond relativization

- **L vs P**

Known: $\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P}$

Open: Does $\mathbf{L} = \mathbf{P}$ hold?

Implications: Negative answer would confirm fundamental limits of space-restricted computation

1.8.2 Specific Separation Problems

- **Space Hierarchy Tightness**

Theorem: $\mathbf{SPACE}(o(f(n))) \subsetneq \mathbf{SPACE}(f(n))$ for space-constructible f

Open: Find natural problems in $\mathbf{SPACE}(n^2) \setminus \mathbf{SPACE}(n)$

Recent Progress: Limited for sub-polynomial separations

- **NL vs L**

Known: **NL** = **coNL** (Immerman-Szelepcsényi)
Open: Does nondeterminism help in logspace?
Approaches: Current attempts focus on branching programs

1.8.3 Space-Time Tradeoffs

- **Fundamental Limits**

Key Question: Can SAT be solved in $n^{1+o(1)}$ space and $2^{n^{o(1)}}$ time?
Barriers: Current techniques cannot rule out mild exponential time

Research Connections These problems relate to:

- Circuit complexity (e.g., **NC** vs **P**)
- Descriptive complexity (logical characterizations)
- Pseudorandomness and derandomization

1.9 The Complexity of TQBF

1.9.1 Basic Properties and PSPACE Membership

Definition 1.9.1 (TQBF). `TQBF` (True Quantified Boolean Formula) is the set of fully quantified Boolean formulas of the form:

$$\Phi = Q_1 x_1 \; Q_2 x_2 \; \ldots \; Q_n x_n \; \phi(x_1, x_2, \ldots, x_n)$$

where:

- Each $Q_i \in \{\exists, \forall\}$ is a quantifier,
- ϕ is a propositional Boolean formula over the variables $x_1, \ldots, x_n$,
- Every variable appears within the scope of its quantifier.

The goal is to determine whether Φ evaluates to true.

Theorem 1.9.1. $\texttt{TQBF} \in \texttt{PSPACE}$.

Proof. We describe a recursive algorithm to evaluate a fully quantified Boolean formula:

1. If Φ has no quantifiers, evaluate ϕ directly under the current assignment
2. For $\Phi = Qx\ \Psi$:
   - Recursively evaluate Ψ with $x = 0$ and $x = 1$
   - Return $\text{Eval}(\Psi|_{x=0}) \vee \text{Eval}(\Psi|_{x=1})$ if $Q = \exists$
   - Return $\text{Eval}(\Psi|_{x=0}) \wedge \text{Eval}(\Psi|_{x=1})$ if $Q = \forall$

Space Analysis:

- Depth of recursion: n (number of variables)
- Space per frame: $O(1)$ (current variable assignment)
- Total space: $O(n)$

Thus, `TQBF` can be decided in polynomial space. □

1.9.2 NP-Completeness of TQBF

Theorem 1.9.2. The restricted version of `TQBF` with only existential quantifiers is `NP`-complete.

Proof. We show ∃-TQBF is `NP`-complete:

Membership in NP:

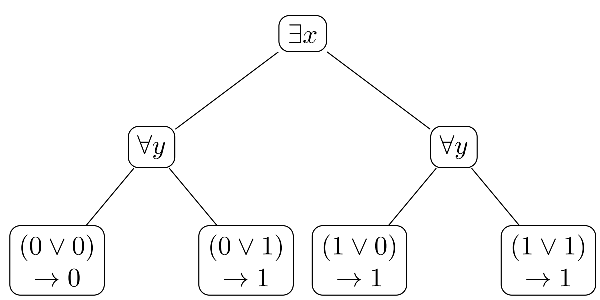
- For $\exists x_1 \cdots \exists x_n \phi(x_1, \ldots, x_n)$:
- Guess values for $x_1, \ldots, x_n$ (polynomial size certificate)
- Verify ϕ in polynomial time

NP-Hardness:

- Standard `SAT` reduces to ∃-TQBF:
- Given CNF ϕ, view it as $\exists x_1 \cdots \exists x_n \phi(x_1, \ldots, x_n)$
- ϕ is satisfiable iff the quantified formula is true

Thus, ∃-TQBF captures the full complexity of `NP`. □

1.9.3 Visualizing the Recursive Evaluation



Result: $\exists x \forall y (x \vee y)$ is true since right subtree satisfies $\forall y$

1.9.4 Space Efficiency

Key observations about the algorithm:

- **Depth-first evaluation**: Only one path active at any time
- **Space reuse**: Each completed subtree's space is reclaimed
- **Constant overhead**: Per recursive call uses $O(1)$ space
- **Total space**: $O(n)$ for n variables

This shows how `TQBF` can be in `PSPACE` despite the exponential recursion tree.

1.10 The Ladder Problem

Definition 1.10.1 (Ladder Problem). The *Ladder Problem* refers to the question: *Given two space bounds $f(n)$ and $g(n)$ where $f(n) = o(g(n))$, does it hold that* $\mathbf{SPACE}(f(n)) \subsetneq \mathbf{SPACE}(g(n))$*?*

In other words, are there problems that require strictly more space than $f(n)$, but can be solved within $g(n)$ space?

Theorem 1.10.1 (Space Hierarchy Theorem (Ladder Theorem)). Let $f(n)$ be a space-constructible function. Then there exists a language L such that:

$$L \in \mathbf{SPACE}(f(n)) \setminus \bigcup_{g(n)=o(f(n))} \mathbf{SPACE}(g(n)).$$

That is, $\mathbf{SPACE}(o(f(n))) \subsetneq \mathbf{SPACE}(f(n))$.

Proof. The proof uses diagonalization. We construct a language L that cannot be computed in space $o(f(n))$, but can be computed in $O(f(n))$ space.

Let $f(n)$ be a space-constructible function. This means we can design a Turing machine that, on input 1^n, uses exactly $f(n)$ space.

We construct a language L that "diagonalizes" against all Turing machines $M_1, M_2, M_3, \ldots$ that run in $o(f(n))$ space.

On input x, the machine D (deciding L) proceeds as follows:

1. Compute $f(|x|)$ (possible due to space-constructibility).
2. Simulate $M_{\#(x)}$ on input x, where $\#(x)$ is the Gödel number of x.
3. If $M_{\#(x)}(x)$ accepts within $f(|x|)$ space, then D rejects; otherwise, D accepts.

This construction ensures that L differs from each M_i on at least one input—specifically, its own index. Thus, L cannot be decided by any machine running in $o(f(n))$ space.

Space Used: All of the above can be implemented within $O(f(n))$ space: computing $f(n)$, simulating a machine up to $f(n)$ space, and inverting its result.

Hence, $L \in \textbf{SPACE}(f(n))$ but $L \notin \textbf{SPACE}(o(f(n)))$.

$\square$

Remark 1. This theorem justifies the term "ladder" because space complexity classes form a proper hierarchy as we increase the available space. That is:

$$\textbf{SPACE}(\log n) \subsetneq \textbf{SPACE}(n) \subsetneq \textbf{SPACE}(n^2) \subsetneq \dots$$

as long as the space bounds are space-constructible and differ asymptotically.

1.11 Savitch's Theorem

Theorem 1.11.1 (Savitch's Theorem). For any function $f(n) \geq \log n$ that is space-constructible,

$$\texttt{NSPACE}(f(n)) \subseteq \texttt{DSPACE}(f(n)^2).$$

Remark 2. Savitch's Theorem shows that nondeterministic space is not much more powerful than deterministic space. In particular, it implies that:

$$\texttt{PSPACE} = \texttt{NPSPACE}.$$

This is in stark contrast with time complexity, where it is widely conjectured that $\texttt{P} \neq \texttt{NP}$.

Proof. We describe a deterministic algorithm that simulates a nondeterministic Turing machine (NTM) using only $O(f(n)^2)$ space.

Let M be a nondeterministic Turing machine that operates in $f(n)$ space on input x of length n. Let C_1 and C_2 be two configurations of M on input x.

We define a recursive procedure:

$$\texttt{REACH}(C_1, C_2, t)$$

which returns true if there is a computation path from C_1 to C_2 in at most t steps.

Base Case: If $t = 1$, return true iff C_2 is directly reachable from C_1 in one step.

Recursive Case: If $t > 1$, we guess a middle configuration C_m and check:

$$\texttt{REACH}(C_1, C_m, \lfloor t/2 \rfloor) \text{ and } \texttt{REACH}(C_m, C_2, \lceil t/2 \rceil)$$

We try all possible C_m. The total number of configurations is at most $2^{O(f(n))}$ since each configuration uses at most $f(n)$ space.

Space Analysis:

- Each recursive call stores C_1, C_2, t, and the guessed C_m, requiring $O(f(n))$ space per call.
- The recursion depth is $O(\log t)$, and $t \leq 2^{O(f(n))}$.
- Therefore, total space is $O(f(n) \cdot \log t) = O(f(n)^2)$.

Thus, a deterministic Turing machine can simulate the NTM using $O(f(n)^2)$ space. □

Remark 3. The key insight is that while time is inherently sequential and expensive to simulate deterministically, space can be reused. Savitch's algorithm uses divide-and-conquer to recursively check for reachability in the configuration graph of the machine.

1.12 The Classes L and NL

1.12.1 Definitions

Definition 1.12.1 (Logarithmic Space Complexity). Let M be a Turing machine with:

- A read-only input tape
- A read/write work tape
- (For nondeterministic machines) A read-once witness tape

We say M operates in *logarithmic space* if it uses at most $O(\log n)$ cells on its work tape(s) for inputs of size n.

Definition 1.12.2 (The Class L). L is the class of languages decidable by deterministic Turing machines in logarithmic space.

Definition 1.12.3 (The Class NL). NL is the class of languages decidable by nondeterministic Turing machines in logarithmic space.

1.12.2 Key Results

Theorem 1.12.1 (Basic Relationships). The logarithmic space classes satisfy:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP}$$

Theorem 1.12.2 (Immerman-Szelepcsényi). $\mathsf{NL} = \mathsf{coNL}$, demonstrating that nondeterministic logarithmic space is closed under complement.

1.12.3 Complete Problems

- L-complete:
  - Undirected s-t connectivity (USTCON)
  - Deterministic acyclic automaton emptiness
- NL-complete:
  - Directed s-t connectivity (STCON)
  - 2SAT (satisfiability of 2-CNF formulas)

1.12.4 Algorithmic Techniques

Example 1.12.1 (Reachability in NL). The STCON problem can be solved by:

Initialize $c \leftarrow s$ (current vertex)
for $i \leftarrow 1$ to n **do**
    Nondeterministically choose neighbor v of c
    $c \leftarrow v$
    if $c = t$ **then**
        return true
    end if
end for
return false

This uses $O(\log n)$ space to track the current vertex.

1.12.5 Open Problems

- Does L = NL? (Widely believed to be false)
- Is NL = P? (Related to whether we can derandomize space-bounded computation)